Debreceni Egyetem
Informatikai Kar
Alkalmazott Matematika és Valószínűségszámítás tanszék

Web 2.0 alkalmazásfejlesztés

Témavezető:
Jeszenszky Péter
egyetemi tanársegéd

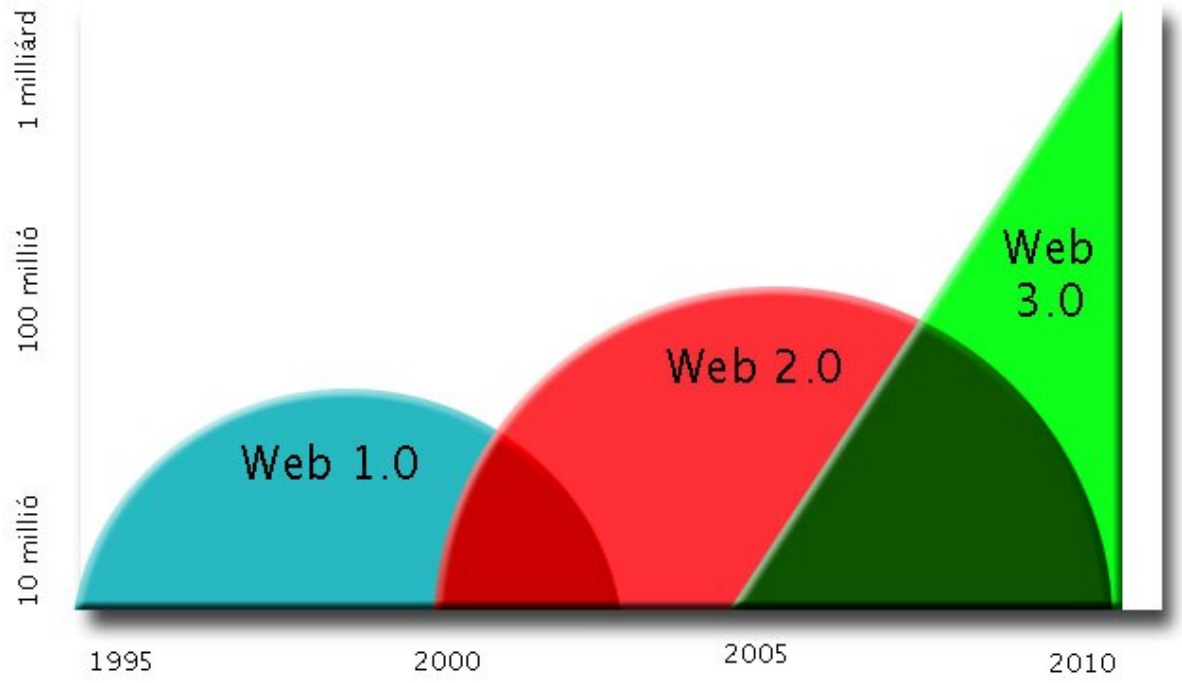
Készítette:
Pogány Tamás
programtervező informatikus

Debrecen
2007

Tartalomjegyzék

1. Bevezetés

Napjainkban az Internet gyorsabban fejlődik, mint eddig bármikor. A hatvanas évek elméleti megalapozása után 1969-ben megjelent az ARPANET (a katonai és tudományos hálózat). 1971-ben már használták az első e-mail programot, majd 1974-ben először használták az Internet kifejezést. 1983-ban az ARPANET-ről leválasztották a katonai szegmenst, ezzel létrejött a ma ismert Internet. Ettől az időponttól kezdve indult el az Internet világhódító útjára. Az e-mail forradalmát követte a hírcsoportok forradalma, majd 1991-ben megszületett a WWW. A rohamos terjedésnek köszönhetően 1995-től már a Web 1.0 korszaka volt (bár a fogalmat csak később, a Web 2.0 idején vezették be), 2000. körül pedig már megjelent a Web 2.0, amely napjainkban éli fénykorát. Ám alighogy megszoktuk az új elnevezést, megállíthatatlanul közeledik (vagy már itt is van) a Web 3.0, de már óvatos jóslásokat lehet hallani a Web 4.0-ról is (1. ábra).



1. ábra

A Web 1.0, 2.0 és 3.0 kialakulása

A Web 2.0-t rengeteg cég tűzte zászlajára. A Google sorra jeleníti meg webes alkalmazásait, melyek lassan alkalmassá válnak az irodai programcsomagok teljes kiváltására. A kezdetben meghívásos rendszerű GMail – ahol a bejutásra hónapokat kellett várni – ma már mindenki számára hozzáférhető. A Google Dokumentumok egy igazi online irodai alkalmazás, de találunk még online RSS olvasót, profi webstatisztikát, mindezt teljesen ingyen. Az internetkapcsolatok gyorssá és megfizethetővé válása hozta magával a videómegosztó portálok megszületését, elsők között a YouTube-ét. Közösségi oldalak jelentek meg (myspace, iwiw), és megszülettek a blogok. Végül, de nem utolsósorban meg kell említenem a Wikipediát, amit az egyik legjelentősebb kezdeményezésnek tartok. A legfőbb cél a tudás megosztása lett, nem annak öncélú birtoklása.

A webalkalmazások sokaságában az embernek akaratlanul is megfordul a fejében: mi lenne, ha én is létrehoznék valamit? Egy jó ötlettel könnyen lehet milliókat keresni, de alkothatunk valamit csak az alkotás szépségéért is. Ebben lehet segítségünkre az Interneten fellelhető számtalan API, de mi magunk is írhatunk, ha nem találnánk megfelelőt. Nap mint nap hallani új dolgok felbukkanásáról a Web 2.0 égisze alatt. Egy részükből sikertörténet lesz, másik részük pedig menthetetlenül eltűnik a süllyesztőben. Hogy az általunk létrehozott webalkalmazás melyik részhez fog tartozni, csak rajtunk múlik.

Dolgozatom témája: „Web 2.0 alkalmazásfejlesztés”, maga az írásmű két részre tagolódik. Az első részben egy Ajax-os galériáról lesz szó. Honlap-üzemeltetőként nagyon nagy szükségem lett volna egy egyszerűen használható, jól karbantartható és elegáns megjelenésű galériára. Ám az Interneten fellelhető számtalan változatok különböző okok miatt nem feleltek meg elképzeléseimnek. Ekkor döntöttem el, hogy sajátot írok. Ennek lassan másfél éve, a galéria azóta rengeteget fejlődött, jelen pillanatban két változat létezik, az 1.1.0-s és a 2.1.0-s verziók. A második részben célom egy Google Maps mashup[1] készítése, amely a debreceni hotspotokat[2] lesz hivatott összegyűjteni. Ez utóbbi megírása mellett azért döntöttem, mert szeretném mélyebben megismerni a Google Maps-et, ugyanis rengeteg lehetőséget látok benne.

[1] A mashup több, létező szolgáltatásnak olyan integrálása, melynek során a szolgáltatások kódját nem változtatják.

[2] A hotspot korlátozás nélkül hozzáférhető vezeték nélküli hálózat.

1.1 Web 1.0, 2.0, 3.0 – mit is takarnak a fogalmak

A WWW kezdetén Web 0.0-ról beszélhetünk. A hálózat már létezik, de nehéz használni, érdemi információ igen kevés van rajta. 1995-től kezdve beszélhetünk Web 1.0-ról. A szájtok száma ekkorra eléri a 250 ezret, a felhasználók száma a 45 milliót. Ez a korszak a „csak olvasható" web időszaka. A felhasználó kereshetett információt, és elolvashatta azt, de a tartalmat nem módosíthatta. Pontosan ezt akarták a webszájtok tulajdonosai: megjelenni az Interneten, a róluk szóló információt éjjel-nappal elérhetővé tenni. Ide tartoznak, bármilyen meglepő, a webáruházak is.

2000. körül kezdte bontogatni szárnyait a Web 2.0. Ez az „írható-olvasható" web korszaka. A webszájtok száma körülbelül 80 millió, a felhasználók száma meghaladja az 1 milliárdot. A felhasználói interakcióra épít, a felhasználók által generált tartalom kerül a középpontba. Megjelennek a blogok, a videómegosztó portálok, közösségi szájtok, webalkalmazások, és nem utolsósorban megjelenik az Ajax.

2005. körül kezdenek a Web 3.0-ról beszélni. „Írható-olvasható-futtatható" web. Elemeiben létezik (ahogy valamikor a Web 2.0), csak még nem állt össze. Jelen vannak az online irodai alkalmazások, a Flash pedig a videómegosztók miatt biztos pozíciót épített ki magának, kvázi videómegosztási szabvánnyá vált. Megvannak tehát a webes operációs rendszer alapjai, komponensei léteznek, használhatóak. Sőt, néhány próbálkozást is láthatunk már WebOS témában: Craythur, Desktoptwo, eyeOS, Xindesk. Jelenleg ezek kísérleti állapotban vannak, nagyrészük 1 GB tárhelyet ad. Terjedésük fő akadályát abban látom, hogy irányukba hiányzik a bizalom. Hiszen fontos fájljainkat nem bíznánk csak úgy egy WebOS-re. Áttörést talán a Google hozhat, amennyiben saját WebOS-t fog fejleszteni.

De a jövő csak ennyiből állna? A Web 3.0 és továbbmenve a 4.0 akkor fog igazi előrelépést hozni, amikor kiszabadul a számítógépek köréből. Gondoljunk az IPTV-re, amelynek segítségével Interneten nézhetünk TV adásokat. Már most lehet Skype telefont kapni körülbelül 35 ezer forintos megfizethető áron, amellyel Skype beszélgetéseket lehet kezdeményezni, csupán egy hotspot szükséges hozzá. És egyre több város csatlakozik a „Netet mindenhová" kezdeményezéshez. Hotspotokat helyeznek el a közlekedési lámpákon, nagyobb épületeken, ezáltal az Internet szinte mindenhol jelen van a városokban. Pár év, és a

jobbnál jobb ötletek robbanásszerű elterjedése lehet, hogy sokkal hamarabb elhozza a Web 4.0-t, mint azt most bárki is gondolná.

1.2 Tanmese

Hogy jobban megértsük, mit is jelentenek ezek a fogalmak a gyakorlatban, idéznék egy mesét, amely eredeti formában az sg.hu-n [6] olvasható.

Sztrogof Mihály, hivatásos futár, megbízást kap minden oroszok cárjától: el kell jutnia Irkutszkba, a Bajkál-tó mellé. A tatárok fellázadtak, valószínűleg ezt a várost is meg fogják ostromolni. Üzenetet kell vinni az ott tartózkodó nagyhercegnek: „Tartsátok a várost, jövünk!" A feladat nem egyszerű, Irkutszk nagyon messze van Szentpétervártól. Át kell kelni a Volgán, a Kámán, meg kell mászni az Urál hágóit, ott a végtelen sztyeppe, a jeges Angara, és a fellázadt tatárok. Vonat csak az út egy részén van, másik részét postakocsin, gyalog, vagy lóháton kell megtenni. Indulás reggel.

Mit kell tennie Sztrogof Mihálynak? Vonatoknak utána járni, postaállomásokat feltérképezni, szállásról gondoskodni, tájékozódni a tatárok elleni hadműveletekről. Meg kell tudni, árad-e a Káma, befagyott-e az Angara, járhatóak-e a hágók.

Szerencsére itt van az Internet. Ha Sztrogof a Web 0.0 világában él, akkor reménye nem sok van használható információt fellelni. Ha az indulási parancsot a Web 1.0 korszakában kapja, akkor csak el kell indítania egy böngészőt, és meglátogatnia a vasúttársaságok, postaállomások, fuvarosok, komposok, meteorológia állomások, stb… honlapját, és az adatok összegyűjtése után magának összeállítani az útitervet.

Ha az indulás még későbbre, a Web 2.0 korszakára esik, akkor még egyszerűbb a helyzet. A hálózat már nem csak linkekkel összekötött dokumentumok halmaza, hanem egyre inkább emberek és szolgáltatások közötti hálózat. Megnézheti a Wikipédiában, hogy mit kell tudni a Bajkál-tóról. Elolvashatja azok blogjait, akik a közelmúltban átkeltek a sztyeppén, sőt, útjáról maga is írhat blogot. Rákeverheti a Google Earth-re a hadijelentéseket, bejelentkezhet közösségi szájtokra, felveheti ismerősei közé a Nagyherceget. Rákereshet a Google-ben a „postakocsi" kulcsszóra, és megnézheti, hogy az emberek mit tartanak legfontosabbnak ebben a témakörben.

Ez már egy sokkal információ-gazdagabb világ, de a sok megtalált adatnak Sztrogof fog értelmet adni, ő rakja össze a képet. De mi a helyzet, ha a tatárok a Web 3.0 korszakában támadnak? Sztrogof leül a gép mellé, és beírja ezt a kérdést: „Hogyan jutok el leggyorsabban Szentpétervárról Irkutszkba?" A gép megérti a kérdést, adatokat és információkat keres, értelmezi és összekapcsolja azokat, gondolkodik Mihály helyett. Végül válaszol a kérdésére: vettem neked jegyet az omszki gyorsra, indulás reggel 7:15-kor. Foglaltam szállást éjszakára, reggelre pedig indulásra készek lesznek a lovak. 2 nap alatt eléred a kompot, amivel átkelhetsz az Angarán, stb… A kérdés csak az, hogy ez a világ mikor, és milyen formában fog eljönni.

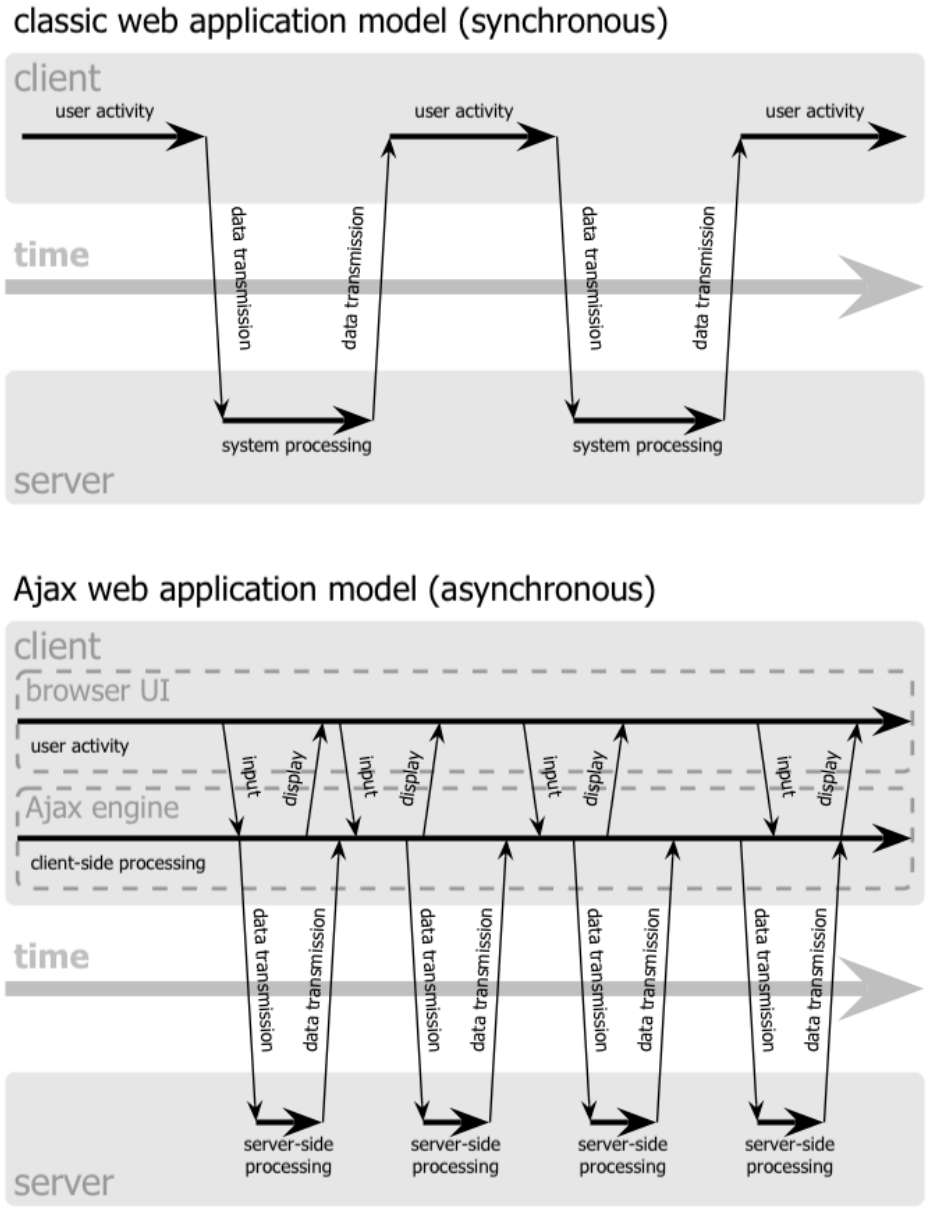
1.3 Összefoglaló táblázat

	Web 1.0	Web 2.0	Web 3.0
Tartalom jellemző megjelenése	Honlapok	Blogok	Szemantikus blogok
Tartalomszolgáltató rendszerek	CMS-ek	Wikipedia	Szemantikus Wikik
Keresők	Altavista, Google	Google	Szemantikus keresők
Online könyvtárak	Gutenberg projekt	Google Scholar	Szemantikus digitális könyvtárak
Közösségépítő lehetőségek	Üzenőfalak	Közösségi szájtok	Szemantikus közösségi szájtok

2. Az Ajax

Mára már egyértelművé vált, hogy a Web 2.0-val megjelenő igények kiszolgálása olyan megoldásokkal lehetséges, amelyek az Ajax technológiára épülnek. Gondoljunk csak bele, mit várunk például egy online szövegszerkesztőtől. Legyen benne automatikus mentés, helyesírás ellenőrzés, formázás. Tudja több ember szerkeszteni a dokumentumot, a módosítások a

közzététel után azonnal látsszanak. Ezek megvalósításához kivétel nélkül Ajax szükséges. De mi is az Ajax? A mozaikszó feloldása: Asynchronous JavaScript and XML, vagyis Aszinkron JavaScript és XML. Az Ajax tehát XHTML, DOM, és XML kombináció, amelyek együttes használatával aszinkron kapcsolat építhető ki a szerverrel. Vagyis a kliens, a webböngésző képes adatot cserélni a szerverrel, és a válasz információt felhasználva a lap egy kis részét frissítheti, anélkül, hogy az egész oldalt újra kellene töltenie (2. ábra). Ez növeli a honlap interaktivitását, sebességét, használhatóságát.



2. ábra

Aszinkron és szinkron kommunikáció a szerverrel. Forrás: [11]

A kifejezést először Jesse James Garrett használta 2005. februári, „A New Approach to Web Applications” című cikkében [11]. Később kijelentette, hogy az Ajax nem betűszó. Ezért találkozhatunk jelenleg is kétféle írásmóddal: AJAX, és Ajax. Az Ajax tehát 2005-ben „született”, azonban komponensei jóval előbb jelen voltak. Az Internet Explorer 1996-ban már támogatta az IFRAME elemet, amely src attribútumának manipulálásával (JavaScript) HTML lapokat tölthettünk be az oldal egy részébe. Volt egy olyan kezdeményezés, amelyben egy Java applet kérte le a szerverről az adatokat. Majd 1997-ben megjelent a jelenleg is használatos XMLHttpRequest, az 5-ös Internet Explorerben. Az XMLHttpRequest egy olyan JavaScript objektum, amely HTTP kliens funkcionalitással rendelkezik, így adatokat tud küldeni szervernek, és adatot tud onnan fogadni. 2002-ben a Java appletet végleg leváltotta az XMLHttpRequest. A JavaScriptet 1997. és 1999. között szabványosították, az XML 1.0-t pedig 1998-ban definiálták. Látható tehát, hogy az Ajax igazából nem új technológia, hanem már meglévők összeolvadásából keletkezett. Az alábbiakban áttekintjük egy egyszerű Ajax-os oldal létrehozását.

2.1 Célkitűzés

Egy XHTML lapra létrehozunk egy gombot. A gomb megnyomásával Ajax kérés fog indulni a szerver felé. A szerver lehetséges válaszai közül mindkét típust (XHTML és sima szöveg, bővebben később) meg fogjuk nézni. A böngésző szabványos Ajax loader-rel[3] fogja jelezni, hogy Ajax kérés van folyamatban. Természetesen tájékoztat róla, ha hiba történt, például a szerver megadott ideig nem válaszolt.

2.2 Az XHTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="hu" lang="hu">
        <head>
                <script src="ajax.js" type="text/javascript"></script>
                <title>AJAX próba</title>
        </head>
        <body>
                <input type="button" value="Click" onclick="loadText()"
                style="margin-right:10px;" />
                <img src="ajaxloader.gif" width="16" height="16"
                stlye="display:none; margin-right:10px" id="ajaxloader" />
```

[3] Az AJAX loader egy animált GIF képfájl, ami az adatok küldését és fogadását jelzi a felhasználónak

```
                <span id="ajaxresponse"></span>
        </body>
</html>
```

Látható, hogy a JavaScript kód az ajax.js fájlban lesz, amelyben kötelezően lennie kell egy loadText() metódusnak. Az Ajax loader display tulajdonságának manipulálásával fogjuk elérni, hogy megjelenjen az Ajax kérés elején, és eltűnjön a végén, illetve az ajaxresponse span elembe fogjuk beilleszteni a szerver válaszát.

2.3 A JavaScript

A JavaScript létrehozása során az első probléma, amivel szembekerülünk a böngészőkompatibilitás. Létre kell hoznunk egy objektumot, amivel Ajax kérést indíthatunk a szerver felé. Internet Explorernél másképp kell eljárnunk, mint a többi böngésző esetén (Safari, Firefox, Opera). Leggyakrabban egy ehhez hasonló megoldást szoktak ajánlani:

```
var ajaxRequest;
try{
        ajaxRequest = new XMLHttpRequest();
} catch (e){
        try{
                ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
                try{
                        ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
                } catch (e){
                        alert("Your browser broke!");
                        return false;
                }
        }
}
```

Ezt a kódot én nem tartom szerencsésnek, mert bár böngészőkompatibilis, de a try-catch blokkok sokasága miatt átláthatatlan. Egy sokkal elegánsabb kódot fogok használni:

```
function g(s) {
        x = 0;
        try {
                x = new ActiveXObject(s);
        } catch (s) {  }
        return x;
};
 […]
var x=false;
 […]
x = window.ActiveXObject ? g('Msxml2.XMLHTTP') || g('Microsoft.XMLHTTP') : new
XMLHttpRequest();
```

Ezzel el is készült az objektum. Ahhoz a ponthoz érkeztünk, amikor el kell döntenünk, a szerver milyen adatokat fog küldeni a kliensnek. Természetes választás az XML, hiszen ez illeszkedik legjobban az Ajax koncepciójához. Használata meggondolandó, ugyanis az adatok kinyerése meglehetősen számításigényes lehet kliens oldalon. Az XHTML válasz akkor célravezető, ha a kódot csak egy az egyben be kell illeszteni az adott helyre, változtatás nélkül. Előnye kliens oldalon az egyszerűség, szerver oldalon azonban többletterhelést okozhat. Létezik egy harmadik lehetőség is, a JSON. Ekkor a szerver egy olyan karaktersorozatot küld, amely könnyen JavaScript objektummá alakítható. Erre a megoldásra nem térek ki külön.

Néhány Mozilla böngésző nem működik megfelelően, ha a szerver válasza nem tartalmaz megfelelő mime-típus fejlécet. Ezért kliens oldalon kell átírnunk a fejlécet. XHTML válasz esetén a szükséges sor:

```
x.overrideMimeType && x.overrideMimeType('text/html');
```

XML válasz esetén:

```
x.overrideMimeType && x.overrideMimeType('text/xml'); .
```

Meg kell mondanunk, hogy melyik JavaScript függvény fogja majd feldolgozni a szerver válaszát:

```
x.onreadystatechange = processChoices;
```

És végül el kell indítanunk a kérést a szerver felé:

```
x.open("GET", url, true);
x.send(null);
```

Az open eljárás első paramétere GET, POST, vagy HEAD. Megjegyzendő, hogy mindig nagybetűvel kell őket írni, ellenkező esetben a Firefox nem fogja végrehajtani a kérést. A második paraméterben kell megadnunk a lekért lap URL-jét. Gyakori hiba, hogy más domain-ről akarunk lekérni lapot. Ez biztonsági okokból nem lehetséges. A harmadik paraméter mondja meg, hogy a kérés aszinkron-e. Ebben az esetben igen, hiszen Ajax kérést hozunk létre. A send paramétereként adhatjuk meg azokat az adatokat, amiket a szerver felé akarunk küldeni. Mi nem akarunk semmit küldeni.

Most meg kell írnunk azt a függvényt, ami a szerver válaszát fogja feldolgozni. Először is vizsgálnunk kell, hogy a szerver válasza teljes mértékben megérkezett-e:

```
if (x.readyState == 4)
```

majd le kell ellenőriznünk a szerver válaszának állapotkódját. Akkor dolgozhatjuk fel a kérést, ha az állapotkód 200 (OK), vagy 304 (Not modified). Gyakori hiba a 304-es státusz kihagyása, hibás működéshez vezethet. A szükséges kód:

```
if (x.status == 200 || x.status == 304).
```

A szerver válasza tehát kétféle lehet. XHTML esetén van könnyebb dolgunk. Ekkor az x. responseText a szerver válaszának szövegével tér vissza, amelyet beilleszthetünk egyetlen utasítással a span elembe:

```
document.getElementById('ajaxresponse').innerHTML=x.responseText;
```

XML válasz esetén a beillesztés korántsem ennyire triviális. Az x.responseXML a szerver válaszát XMLDocument objektumban adja vissza, amit DOM függvényekkel dolgozhatunk fel. Természetesen minél bonyolultabb az XML, annál erőforrásigényesebb a feldolgozása. Ebben a példában a szerver egy egyszerű válasszal fog visszatérni. Így tudjuk feldolgozni:

```
var xmldoc = x.responseXML;
var root_node = xmldoc.getElementsByTagName('root').item(0);
document.getElementById('ajaxresponse').innerHTML=root_node.firstChild.data;
```

2.4 A szerver oldal

Szerver oldalon PHP-t fogunk használni. XHTML válasz esetén a dolgunk rendkívül egyszerű: <?php echo "Hello"; ?>. Míg XML esetén a kód a következő:

```
<?php
        echo "<?xml version=\"1.0\" encoding=\"iso-8859-2\" ?> ";
        echo "<root>";
        echo "Hello";
        echo "</root>";
?>
```

3. Ajax galéria

Eddigi munkáim során sok galériával volt dolgom, mindegyik rendelkezett kisebb-nagyobb hibával, hiányossággal. Az adatbázis alapúak alapvető problémája, hogy kezelésük nehéz. Mindenképpen szükség van egy adminisztrációs felületre, ahonnan felvehetjük az adatbázisba az új képeket. Törlés ugyanezen a felületen történik. Egy-egy könyvtár feltöltése indokolatlanul időigényes. Egyetlen alternatíva tehát a fájlműveleteken alapuló galéria. Azonban ez a módszer is felvet néhány kérdést.

Természetes igény a magyar ékezetes karakterek korrekt támogatása. Megvizsgálva az alapvetően szükséges PHP függvényeket (adott könyvtárban lévő fájlok beolvasása) látható, hogy a dir könyvtár osztály nem támogatja az ékezetes karaktereket a könyvtár nevében. Kézenfekvő tehát, hogy a könyvtár neveket nem használhatjuk az albumok neveként, másik megoldást kell keresni. A fenti megszorítást figyelembe véve ideális megoldásnak tűnt a következő: az album könyvtárában létre lehet hozni egy info.txt állományt. Amennyiben ez létezik, akkor a benne lévő szöveg jelenik meg az album neveként, egyébként pedig a mappa neve. Természetesen valamilyen korlátot kell szabni a fájlban lévő szöveg hosszára nézve. Előzetes tesztek alapján a 40 karakter jó megoldásnak tűnt, és a tartós használat igazolta a jó választást. Meg kellett még akadályozni, hogy bármiféle kódot lehessen injektálni ebbe a fájlba. Figyelmetlen jogosultság-beállítással (például az extra.hu a fájlokat alapból 666 jogosultsággal hozza létre) ez a fájl írható, így kívülről manipulálható lehet. Erre megoldás a PHP htmlspecialchars függvénye, amely megakadályozza a kódinjektálás az által, hogy bizonyos speciális karaktereket átkonvertál.

Biztosítani kell még a magyar karakterek megfelelő megjelenítését az Ajax válasz feldolgozása során is. Jó megoldás az urlencode és urldecode függvények használata. Szerver oldalon a válasz szükséges részeit (a magyar karaktert tartalmazókat) urlencode függvénnyel kódoljuk, majd a választ kliens oldalon dekódoljuk. JavaScriptben szükséges függvények az escape és az unescape. Azonban a kezdeti tesztek során nyilvánvalóvá vált, hogy a PHP urlencode és a JavaScript unescape függvénye képtelenek együttműködni, a kapott eredmény meg sem közelíti a vártat. Például dekódolás után szóközök helyett + jelek maradtak. Kliens oldalon tehát saját dekódoló függvényre volt szükség. Erre a célra az Interneten rengeteg helyen fellelhető, szabvány kódot használtam:

```
function URLDecode(encoded)
{
	var HEXCHARS = "0123456789ABCDEFabcdef";
	var plaintext = "";
	var i = 0;
	while (i < encoded.length) {
		var ch = encoded.charAt(i);
		if (ch == "+") {
			plaintext += " ";
			i++;
		} else if (ch == "%") {
			if (i < (encoded.length-2)&&
			HEXCHARS.indexOf(encoded.charAt(i+1)) != -1&&
			HEXCHARS.indexOf(encoded.charAt(i+2)) != -1 ){
			plaintext += unescape( encoded.substr(i,3) );
				i += 3;
			} else {
				alert( 'Bad escape combination near ...' + encoded.substr(i) );
				plaintext += "%[ERROR]";
				i++;
			}
		} else {
			plaintext += ch;
			i++;
		}
	}
	return plaintext;
};
```

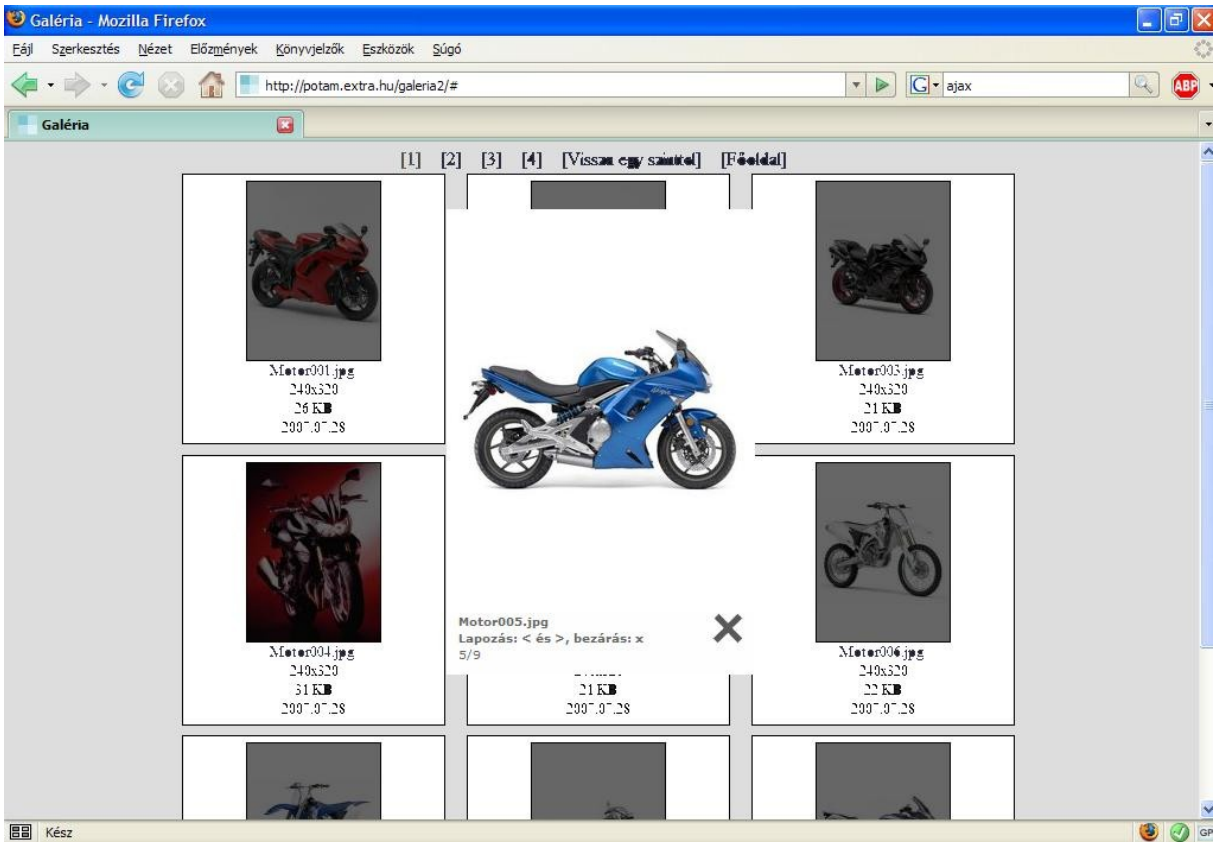
Ez a kód majdnem tökéletes eredményt szolgáltat, kivétel az ő és az ű betű. Tehát a függvény meghívás előtt replace függvénnyel vissza kell kódolnunk ezt a két betűt. Kódjaik: ő - %F5, Ő - %D5, ű - %FB, Ű - %DB. Vigyázni kellett arra a csere során, hogy az összes előfordulást cseréljük, ne csak az elsőt. Ez a probléma nem is annyira triviális, sok magyar Ajax-os próbálkozásnál ez a rész el van rontva. Tehát egy helyes csere például: result=result.replace(/%DB/g,'Ű'); . A „g” jelenti a globális cserét, azaz a szöveg egészében cserélünk.

El kellett még dönteni, hogy a szerver milyen módon fogja küldeni a választ. Tekintettel arra, hogy a szerver mindenképpen bejárja a választott könyvtárat, tehát nem lesz kevesebb munkája egyik módszer esetén sem, így az XHTML válasz mellett döntöttem. A kliens terhelése ezáltal minimálisra csökkent, a választ ugyanis csak dekódolni kell, majd beilleszteni a kívánt helyre.

3.1 Dizájn

Dizájn szempontjából az egyszerűség elvét követtem. Legyen annyira egyszerű, amennyire csak lehet, de mégis elegáns. Legyen érvényes XHTML 1.0 Strict, és ésszerűen legyen felépítve. Azaz a táblázat legyen táblázat, és ne töltsön be térbeli elrendező szerepet. Nagyon sokan esnek ebbe a hibába, holott a CSS2 megjelenése óta idejétmúlt és szükségtelen technikává vált, hogy az egész HTML oldal egyetlen nagy táblázat, és cellái töltenek be strukturális elrendező szerepet. A div elemek megfelelő pozícionálása a követendő minta. A képek alatt jelenjen meg némi információ, minimálisan elvárható a kép létrehozási dátuma, a mérete, és az elfoglalt helye. Követendő mintának tartom a pyxy galéria [12] kinézetét, amely megfelel ezeknek az elvárásoknak. Ugyanakkor hiányossága, hogy nem teljesen Ajax-os, így nagyon sok időbe tellett volna ezt a galériát átalakítani. Végül mintaként felhasználva létrehoztam egy saját dizájnt, szürke színvilággal (3. ábra), Ajax loaderekkel (szintén hiányzik a pyxy galériából).

A képek betöltéséhez Lightbox JS-t használok [13]. Ezt a szkriptet kifejezetten képek Ajax-os megjelenítésére írták, így ideális választás galériához. Néhány helyen kellett csak módosítani. Egy esetben fordítani kellett, másik esetben pedig letiltottam azt a lehetőséget, hogy az oldal hátsó, elsötétedő részére kattintva be lehessen zárni a képet.



3. ábra

Ajax galéria - dizájn

3.2 Egyéb alapkövetelmények, megoldások

Az egész galériának természetesen maximális mértékben testreszabhatónak kell lennie. Ezt a célt szolgálja a config.php, ahol számos egyéni beállítási lehetőség van. Beállíthatjuk azt, hogy melyik könyvtárban van a galéria. Az albumok (vagy mappák) alapértelmezett képét szabadon állíthatjuk. Beállítható az úgynevezett mappakép is. Ha egy ilyen nevű fájl van az album könyvtárában, akkor azt fogja megjeleníteni az album képeként, nem pedig az alap képet. Beállíthatjuk, hogy hány kép legyen egy sorban, és egy lapon maximálisan hány sor legyen. Állíthatunk főoldalra mutató linket, és ki/be kapcsolhatjuk azt.

Beállítható úgynevezett rejtő karakter is. Az ilyen karakterrel kezdődő fájlokat és könyvtárakat a galéria nem veszi figyelembe. Mondhatnánk, hogy mi szükség van erre, a ’.’ karakter tökéletesen megfelel erre a célra. Korántsem ilyen egyszerű a helyzet. Az egyik szerveren, ahol az általam készített galéria üzemel több száz képpel, nem hozható létre ’.’-al kezdődő fájl/könyvtár. Meglepő, de ezt a helyzetet is tudni kell kezelni. Így a ’.’ helyett

tetszőleges karakter megadható, alapértelmezett a ’-’. Gyakorlati haszna van ennek a funkciónak, ugyanis ha a feltöltés várhatóan sok időt vesz igénybe, akkor nem szeretnénk, hogy a felhasználók félkész albumot lássanak. Ekkor egyszerűen az album könyvtárát ’-’-el kezdődő névvel hozzuk létre. Feltöltjük a tartalmat, majd utána tesszük láthatóvá az albumot, vagyis a könyvtár nevének elejéről eltávolítjuk a ’-’ karaktert.

A galériának tudnia kell létrehozni a képek mintaképeit (thumbnail-jeit), amennyiben azok még nem állnak rendelkezésre. Erre a célra a minishowcase [14] mintakép készítő fájljának egy kicsit módosított változatát használom. Képes a GD (a PHP grafikai könyvtára) verzió automatikus felismerésére, és ennek megfelelően létrehozni a mintaképeket. Megjegyzendő, hogy néhány 2–3 megabájtos képeknél már 30–40 mintakép készítése is túllépi a PHP memórialimitjét, így érdemes ilyen esetekben egyénileg létrehozni azokat. Természetesen állítható a mintaképek mérete, és a könyvtáruk neve is.

Mivel a PHP szkript fájlokkal dolgozik, ezért biztosítanunk kell azt, hogy csak és kizárólag azokhoz a könyvtárakhoz férjen hozzá, ahol az albumok vannak. Alaphelyzetben van egy galleries könyvtár (természetesen változtatható), és ebben kell létrehozni az albumok könyvtárait. Az Ajax kérés egy könyvtár betöltése lesz, amit paraméterként megkapva a PHP szkript visszaadja a beillesztendő kódot. Abban az esetben, ha a paraméter kívül esik a galleries könyvtáron, azaz a használat nem szabályos, valaki hatáskörön kívül eső fájlokba akar betekintést nyerni, a PHP szkript üres választ fog küldeni. Ezt a

```
if(preg_match("/^".str_replace("/","",$gallery_dir)."/i",$request_directory)) {
```

kód teszi lehetővé.

Biztosítanunk kell még, hogy a galéria kikapcsolt Register_Globals (globális változók engedélyezése) mellett is működjön. Ezt általában ki is szokták kapcsolni, biztonsági okokból. A problémát a PHP kód elején elhelyezett `import_request_variables("gP","request_");` sor oldja meg. Így hozzáférhetővé válnak a globális változók kikapcsolt Register_Globals mellett is ’request_’ prefix-szel.

3.3 Szerver oldal

Szerver oldalon a legfontosabb PHP szkript a galeria.php. Ezt a szkriptet hívja meg a kliens az Ajax kérés küldésekor, így értelemszerűen ez fogja a választ is küldeni, valamint ez

a szkript hívja meg a thumbnail.php-ben található createThumbnail függvényt is (amennyiben az szükséges).

A betöltendő könyvtár nevét (amit a klienstől kaptunk) a $request_directory tartalmazza, még akkor is, ha a Register_Globals ki van kapcsolva (lásd az előző részt). Legelső dolog természetesen azt ellenőrizni, hogy a könyvtár a megengedett „burkon" belül van-e, minden más művelet csak azután következhet. A szkript egy dinamikus tömböt használ a fájlok és könyvtárak tárolására. A könyvtárat listázni kell, ehhez a PHP dir osztályát, annak is read metódusát használjuk. Fontos leszögezni, hogy a dir nem képes ékezetes karaktereket kezelni, ezért ezek használatát mellőznünk kell. Könyvtár objektum létrehozására a

```
$mydir = dir($request_directory."/");
```

utasítást, bejárására a

```
while(($file = $mydir->read()) !== false)
```

ciklust használom.

Meg kell vizsgálnunk, hogy a bejárás során éppen aktuális $file könyvtár-e, vagy pedig fájl. A PHP is_dir függvénye igaz értékkel tér vissza, ha a paramétere könyvtár, hamissal egyébként. Ha könyvtár, akkor csak abban az esetben vesszük fel a tömbbe, ha nem '.'-al kezdődik, vagy a rejtő karakterrel, vagy nem a thumbnail könyvtárról van szó. Fájlok esetén pedig vizsgáljuk a kiterjesztést (csak jpg-vel dolgozunk), illetve hogy nem az album képéről van-e szó. A szükséges három sor:

```
$pathinfo = pathinfo($file);
$ext = $pathinfo["extension"];
if ((strtolower($ext) == "jpg") && ($file!==$menufile)) {$files[]=$file;}
```

Végül a könyvtárat le kell zárnunk a close metódussal. A kódnak ezen a pontján a files tömbben rendelkezésre áll a betöltendő könyvtárban lévő összes kép és alkönyvtár. A sorrendjük azonban korántsem biztos, hogy megfelelő, ezért a tömböt rendeznünk kell, mégpedig természetes rendezéssel: usort($files, 'strnatcmp');. A természetes rendezés az elemeket számértékük alapján rendezi, így ez a rendezés adja a felhasználó által elvárt eredményt.

3.4 Lapozás

Teljesen nyilvánvaló, hogy a felhasználók egy-egy albumba nagy mennyiségű képet szeretnének feltölteni. Ezért a galéria tetején lapozás linkeket kell biztosítani, praktikusan az oldalszámokat. A lapozás érdekében a szkript paraméterként nem csak a könyvtárat kapja meg, hanem egy alsó (min.) és felső (max.) értéket is. Az ezek által meghatározott intervallumban lévő képek fognak a kimenetre kerülni. Világos, hogy ezeknek az értékeknek alsó és felső korlátot kell szabni, és ha azt túllépik, akkor korrigálni kell. Alsó korlátnak megfelel az 1, felső korlát pedig a files tömb mérete, amit a count függvénnyel kaphatunk meg. Tudnunk kell, hogy hány lapunk lesz. Ez egyszerűen meghatározható, ugyanis tudjuk hány fájlunk és könyvtárunk van, valamint azt is, hogy egy lapon hány sor lesz, és egy sorban hány kép (a config.php-ben állítható). Tehát rendelkezésre áll immár a lapok száma. Lapozás linkeket csak akkor kell kitenni, ha egynél több lapunk van. CSS kóddal gondoskodunk arról, hogy az aktuális lap linkje eltérjen a többi színétől. A lapozás linkek Ajax kéréseket fognak indítani, amik lekérik ugyanazt a könyvtárat, és a min. és max. értékek megfelelő megadása biztosítja a lapozás működését. Azaz ha ’x’ lapra van szükség, akkor egy ’i’ ciklusváltozót növelünk ’1’-től ’x’-ig. Az egy lapon lévő képek száma legyen ’y’. Ebben az esetben az ’i.’ linknél a minimum érték ((i-1)*y+1), a maximum pedig i*y.

3.5 Visszalépés link

Az albumok a könyvtárstruktúrának megfelelően rendeződnek, ennek megfelelően egy album tartalmazhat al-albumokat, és így tovább. A galéria főkönyvtárán kívül ezért szükséges egy visszalépés link, amely a szülőkönyvtárban lévő albumot fogja betölteni. A link természetesen egy Ajax kérést fog indítani, paraméterként a szülő könyvtárat adva meg a galeria.php-nek. Az aktuális könyvtárból a szülőkönyvtár kinyerése egyszerű ötlet alapján történik. Megkeressük az aktuális könyvtárban (mint sztringben) hátulról indulva teljes kereséssel a legelső könyvtár-elválasztó jelet. Annak érdekében, hogy a szkript Windows rendszeren is működjön, egyaránt keressük a ’\’ és a ’/’ jeleket. Ha megvan a pozíciója, akkor az aktuális könyvtár sztringjéből részsztringet képezünk eddig a pozícióig, és az így kapott részsztring lesz a szülőkönyvtár. Szükséges még egy főoldalra mutató link (amely természetesen kikapcsolható), ugyanis a galéria legtöbbször nem önmagában létezik, hanem egy honlap szerves részeként.

3.6 A képek

Az egyes képek és leírásuk külön-külön táblázatokban jelennek meg. Ez ellentmond a korábban említett elvnek, miszerint táblázatot nem szabad strukturális elemként használni. Itt pedig mégis erről van szó, ennek oka a böngészőkompatibilitás. A CSS2 már alkalmas azonos magasságú, és azonos igazítású div-ek készítésére. Ezt a display tulajdonság table, table-row, és table-cell értékei teszik lehetővé. Ezekkel teljes mértékben megfelelő megjelenítést lehet elérni a böngészők nagy részénél. Az Internet Explorer azonban sajnálatos módon nem támogatja a display tulajdonság fent felsorolt értékeit. Ez rendkívül furcsa, ugyanis a dolgozat írásakor az elérhető legfrissebb CSS verzió a hármas, és még a legújabb, 7-es Internet Explorer sem támogat egy CSS 2-ben definiált értéket. A div-ek használatáról tehát le kellett mondanom, és helyettük egycellás táblázatokat használni.

A képek alatt információként megjelenik a méretük, elfoglalt tárhelyük, és a létrehozásuk dátuma (amely azonos a feltöltési idejükkel, 4. ábra). Egy kép mérete egyszerűen lekérdezhető a getimagesize függvénnyel. Ez egy tömböt ad eredményül, aminek az első eleme a kép szélessége, második a magassága. A kép (és akármelyik fájl) méretét bitekben kifejezve a filesize függvény adja vissza, amit az alábbi függvénnyel lehet olvasható formára hozni:

```
function resize_bytes($size)
	{
		$count = 0;
		$format = array("B","KB","MB","GB","TB","PB","EB","ZB","YB");
		while(($size/1024)>1 && $count<8)
		{
			$size=$size/1024;
			$count++;
		}
		$return = number_format($size,0,',','.')." ".$format[$count];
		return $return;
	}
```

4. ábra

Információk a képről

A könyvtárakra mutató linkek megfelelő Ajax kérések lesznek, a képeknél pedig lightbox-ra[4] mutató linkek. Természetesen mindenféle Ajax művelet előtt gondoskodni kell a megfelelő betöltés animáció megjelenítéséről, illetve a válasz feldolgozása után annak eltávolításáról.

3.7 Objektum orientáltság

Még nem esett róla szó, de fontos megemlíteni: a galéria objektumorientált. A PHP-ban lehetőségünk van objektumorientált módon programozni, és a galériánál magától értetődik az osztályok használata. Egyetlen fontos dologra kell figyelnünk a tervezés során: a PHP-ban nincs beépített destruktor függvény. Így ezt nekünk kell kézzel létrehozni, és az objektum megsemmisítése előtt metódusként meghívni. A destruktor függvény tipikusan unset utasítások sorozata, amelyekkel felszabadítjuk az osztály adattagjait. A legfontosabb dolgok PHP objektumorientált programozásnál:

Osztály létrehozása	class a { … }
Példányosítás	$b = new a;
Adattagok	var $adattag;
Objektum adattag elérése	$b->adattag
Konstruktor	Az osztály nevével megegyező nevű függvény

[4] A lightbox a képekre kattintás után megjelenő, előtérbe hozott komponens, ami a képet tartalmazza

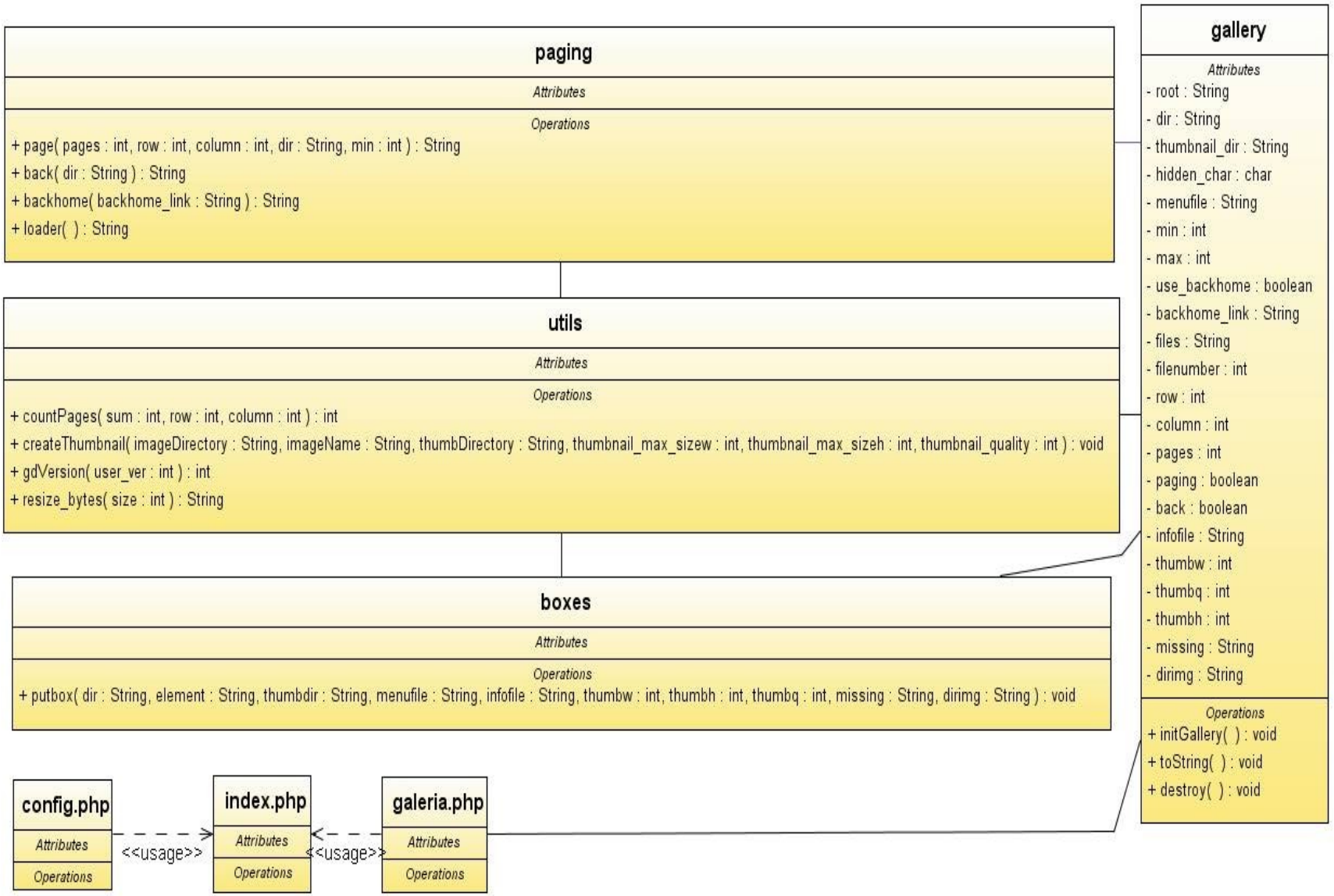
This használata	Pl. $this->adattag
Módszerek	Mint a sima függvények
Módszerek meghívása	$b->módszer();
Destruktor függvény helyes meghívása	$q = new p; $q->destroy(); unset($q);
Kiterjesztés	class s extends p { … }
Statikus módszer meghívása	'::' minősítővel

Mivel a PHP nem rendelkezik csomag fogalommal, ezért ha egy adott fájlból másik fájlban lévő kódot kívánunk meghívni (ebben az esetben egy osztályt példányosítani), akkor require vagy include függvényt kell használni a kód elején. Mindkettő paramétere egy fájl, azonban alapvető különbség van köztük. Az include használatakor a fájl behívása akkor történik meg, amikor a szkript futásakor az értelmező „odaér". Ezzel szemben a require használatakor az értelmező még a kiértékelés előtt behívja a fájlokat. Hiba szempontjából az include csak figyelmeztetés szintű hibát generál, a require függvény pedig fatális hibát. Kézenfekvő tehát, hogy a konfigurációs fájl (config.php) betöltésére az include a megfelelő, hiszen annak hiánya esetén is biztosítható valamilyen szintű működés. Míg az osztályok fájljai esetén a require a kívánatos, hiszen ha nincs meg a fájl, akkor nem tudunk példányosítani, így a szkript futását meg kell szakítani. Erre csak a require alkalmas, az include nem. További finomítás a require_once, és az include_once használata. Ezek a függvények csak egyszer hívják be a paraméterül kapott fájlt, így összefüggő, nagy kódok esetén is hiba nélküli kódot eredményeznek.

Az osztályok megtervezésekor fontos szempont volt az ésszerűség. Az osztályoknak egyszerűeknek, maguktól értetődőknek kellett lenniük. Jó választásnak tűnt a galéria egy lapja, mint egy osztály. Ez a gallery osztály, és a galeria.php fogja példányosítani. A példányosítás után meghívjuk a gallery osztály toString metódusát, amely az Ajax választ fogja tartalmazni. A válasz kiírása után az objektum megsemmisíthető. Mivel a galéria teljes mértékben testreszabható a config.php fájlon keresztül, ez azzal a hátránnyal járt ezen osztály esetén, hogy rengeteg (egészen pontosan 22) adattagot tartalmaz. Összehasonlításképpen, ha semmit nem lehetne a galériában testreszabni, akkor ez a szám 3 lenne. Jó megoldás volt a felső linkek (főoldal, egy szinttel vissza, lapozás), és a képek megjelenítéséért felelős kódokat külön osztályokba sorolni (paging és boxes osztályok). Külön osztályba kerültek a technikai

jellegű függvények, úgymint a countPages, createThumbnail, gdVersion, resize_bytes és search_last_backslash. Ez a utils osztály. Mindhárom előbb említett osztály kiszolgálóosztály. Ez a utils osztály esetén funkciójából adódóan egyértelmű, a másik két osztály esetén pedig nem láttam értelmét példányokat létrehozni egyszerű kiírások miatt.

Tehát összességében a galéria szerver oldalának magját négy osztály alkotja: gallery, utils, boxes, paging. Ebből az utolsó három kiszolgálóosztály. Az osztálydiagram az 5. ábrán látható.



5. ábra
A galéria osztálydiagramja

3.8 Licensz

A licensz megválasztásánál alapelv volt „Az emberi tudás az egész világ tulajdona.” mottó. Ennek megfelelően olyan licenszet kellett keresni, ami biztosítja a galéria szabad és díjmentes felhasználását (beleértve módosítások létrehozásának lehetőségét), de ugyanakkor az eredeti szerzőt nem hagyja feledésbe merülni. A Creative Commons számos szabad licenszet kínál, közöttük könnyen megtalálható a megfelelő. A licenszek 4 alapvető elem szabad összeválogatásából születnek: „nevezd meg”, „ne add el”, „ne változtasd”, „így add tovább”. Ideális választás volt a „nevezd meg”, „ne add el”, „így add tovább” hármas. Így a galéria szabadon terjedhet, de biztosítva van az eredeti szerző nevének feltüntetése, a módosításokért nem kérhető pénz, illetve a módosításokat is a fenti licensszel kell továbbadni. A licensz a következő címen érhető el: http://creativecommons.org/licenses/by-nc-sa/3.0/deed.hu .

3.9 Elérhetőség

Az Ajax galéria mindenkori legfrissebb változatai megtalálhatók a honlapomon: http://potam.extra.hu/large.php?oldal=1 . Ezen az oldalon linkek is találhatók működő demókra. Az 1.x-es verziók abban különböznek a 2.x-esektől, hogy korábbi lightbox verzióra épülnek, így nincs a lightboxban képek közötti lapozási lehetőség. A motorjuk teljes mértékben megegyezik.

3.10 Összefoglalás

A másfél évvel ezelőtt írt galéria mára nagy változáson ment keresztül. Több major verzióváltás után mai formájában megfelel a kor igényeinek. Objektumorientált, kezelése egyszerű, használatba vételi ideje minimális, teljesen testreszabható, így a hozzáértők könnyen módosíthatják. Az Interneten fellelhető Ajax galériák között megállja a helyét, és megfelelő mértékben támogatja a magyar ékezetes karaktereket, így nem jár lemondásokkal. Fájlműveleteken alapul, ez könnyű módosítást, dinamikusságot tesz lehetővé. Kis befektetéssel alkalmassá tehető többnyelvűségre is. Stabil, böngészőkompatibilis, még nem találtam olyan (grafikus) böngészőt, ahol ne működött volna.

4. Google Maps mashup

4.1 A Google

A kilencvenes évek közepén Internet-lázban égett a világ. Az elérhető adatmennyiség olyan nagyra duzzadt, hogy természetes igény mutatkozott a tartalom indexelésére. Szabad volt az út a keresők megjelenéséhez. Tucatszám jelentek meg próbálkozások ezen a téren, a legismertebbek: Lycos, Infoseek, Excite, Altavista. Ezek többsége azonban hamar tönkre is ment, mert nem bírták a kiélezett versenyt. Ebben az időben iratkozott be a Stanford Egyetemre Larry Page és Sergey Brin, akik egy új Internetes keresőn kezdtek el dolgozni. Kész munkájukat aztán értékesíteni próbálták a piacon jelen lévő keresőt üzemeltető cégeknek. Noha az ő algoritmusuk 20%-al jobb eredményt produkált a meglévő keresőknél, nem akadt rá vevő. A cégek azt mondták, hogy bőven elég a 20%-al kevesebb is. Az idő megmutatta, hogy bizony nem volt elég. Így 1999-ben kényszerből 100.000 dollár kezdőtőkével megszületett a Google. Hamar kiemelkedett versenytársai közül, és mára már a legismertebb keresővé vált. Piaci részesedése 60% fölött van. Az oldalak indexelésekor, és sorba rendezésekor egyedi, pagerank-on alapuló algoritmust használ. Azt veszi alapul, hogy hányan hivatkoznak az adott oldalra. Minél többen, annál előkelőbb helyet kap az oldal a találati listán. Természetesen az algoritmus korántsem ilyen egyszerű, azonban a Google üzleti titkait sértené, ha bővebben kifejtenék a technikát. Sikerük abban áll, hogy a keresés eredménye az összes kereső közül náluk áll a legközelebb a felhasználó által elvárthoz. Fontos még a gyorsaság is, az első oldalnyi találatot a kereső személy nagyon gyorsan megkapja.

A cég ismertsége 1999. óta folyamatosan felfelé ível, 2004-ben részvényeket is bocsátottak ki. Szerte a világon vannak kirendeltségei a szilícium völgyi cégnek, és Google dolgozónak lenni nagy megbecsüléssel jár. Olyan nagy emberek dolgoznak ott, mint Wint Cerf, az Internet atyja. A Google bevétele a reklámokból származik, vagyis a fizetett hirdetésekből. Azonban a Google megtanulta, hogyan termelhet profitot az etikai elvek feladása nélkül. A Google ma már több mint kereső, fogalommá vált. Briliáns üzletpolitikájukkal, okos felvásárlásokkal internetes szolgáltatások sorait nyújtják. Övék a YouTube, a Blogger, a Picasa, és még sok más. Különféle szolgáltatásaik száma 40 felett van [27]. De nem csak az

Interneten vannak jelen. Desktop alkalmazásaik is széles körben ismertek, és hamarosan megjelenik a Google Phone, ami a 100 dolláros laptopok vetélytársa lehet.

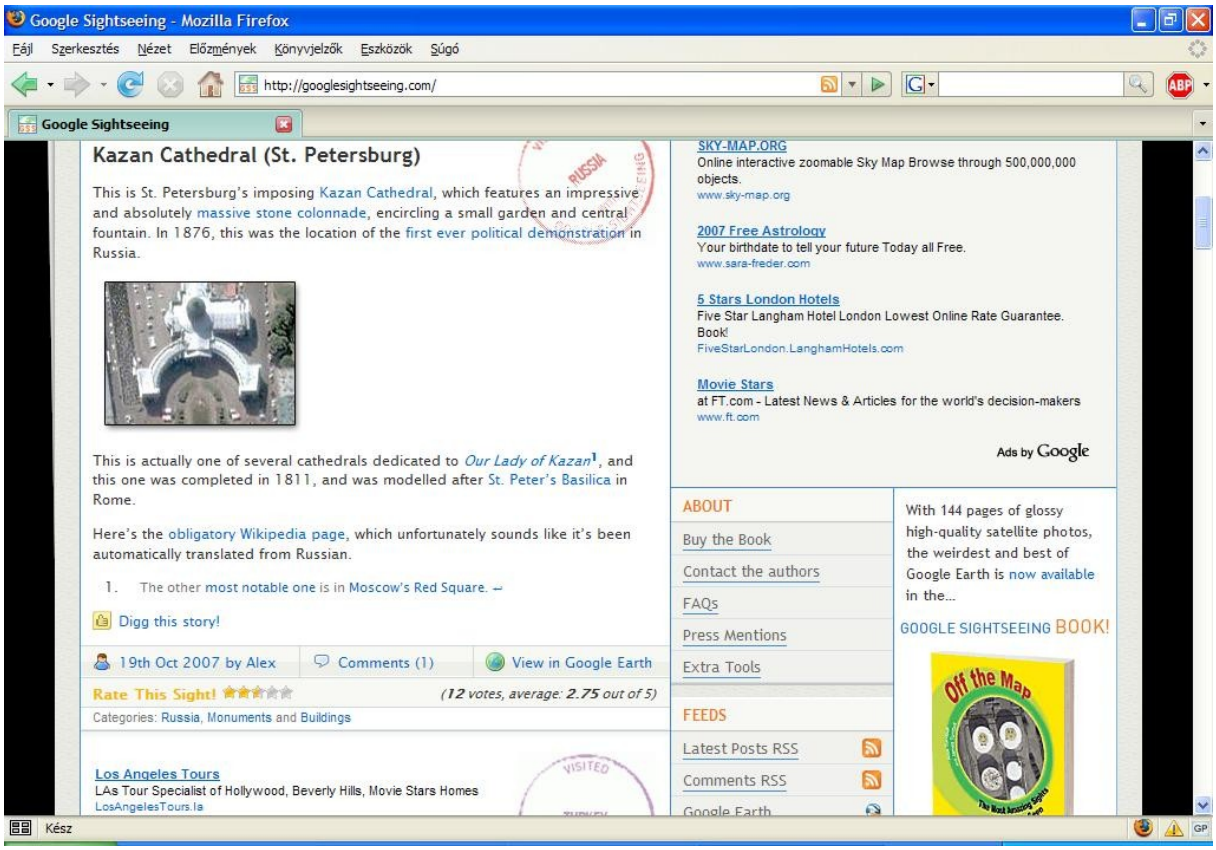
A Google gondol a programozókra is. Sok szolgáltatásához elérhetővé tett API-kat, szabad kezet adva a programozóknak a kiteljesedésre. De nem szabad megfeledkeznünk a nemrég indult forráskód-kereső szolgáltatásukról sem. Odafigyelnek arra, hogy szolgáltatásaik minden felhasználói réteg számára nyújtsanak valamit, kezdve az 1.0-s usertől, a rendszergazdákon át a programozókig.

A Google tehát lassan életünk minden területét behálózza, joggal tekinthető a Nagy Testvérnek. Miért bíznak benne mégis az emberek? A válasz a Google etikai elveiben keresendő. Ha adatokat gyűjtenek, azt sosem azért teszik, hogy az egyénről alkossanak képet, ahogy például a reklámügynökségek teszik. Minden egy nagy közösbe megy, és tömeges adatokat vizsgálnak. Ilyen módon jött létre például a népszerű helyesírás javító, ami hosszú gyűjtőmunka eredménye. Személyes adatokat pedig nem adnak ki senkinek. 2007-ben történt, hogy Németország követelte a GMail felhasználók adatainak kiszolgáltatását. A Google közölte, inkább kivonul Németországból, minthogy ezt megtegye, a Nagy Testvértől való félelem tehát megalapozatlan. A Google sok területen megkönnyítette, és meg fogja könnyíteni az életünket. A bizalomra pedig bőven rászolgáltak az elmúlt évek során.

4.2 A Google Maps

2005. február 8-án jelentette be a Google ingyenes térkép-szolgáltatását, a Google Maps-et. Induláskor a térkép csak az USA, Kanada, Írország, Egyesült királyság területét fedte le. Október 6-án kilépett a béta tesztidőszakból, támogatta az Internet Explorer, Mozilla, Opera és Safari böngészőket. A fejlesztések sokaságát ma már igen nehéz felsorolni. A Google Maps térképei egyre részletesebbek, és ma már nemcsak a Föld, hanem a Hold és a Mars is elérhető az ismert felületen. Nem is olyan régóta (2007. május) utcaszinten is kereshetőek magyarországi címek. Egyik nagyon népszerű újításuk a streetview: bizonyos városok utcáit 360 fokos fényképezőgéppel autókból lefényképezik, így olyan, mintha ténylegesen azon az utcán sétálnánk. A Google Maps mai napig nagy vihart kavar. Személyiségi jogoktól kezdve a katonai kérdésekig sok mindent felvet a létezése. Természetesen vannak kifejezetten erre rájátszó honlapok (6. ábra), ahol az emberek a Google Maps-en talált érdekes képeiket

küldhetik be, legyen az egy nem dolgozó munkás valamelyik város streetview nézetén, vagy tetőn napozó emberek.



6. ábra

A Google Sightseeing oldal

A Google Maps-ben rengeteg lehetőség rejlik, ezt felismerve adta ki a Google szolgáltatásához az API-t. Kezdetben használata rendkívül körülményes volt. Egy kulcsot kellett hozzá igényelni, amely kizárólag egy domainhez volt jó, így a fejlesztés nehézkessé vált. 2007 áprilisában jelentették be a MyMaps szolgáltatást. Azt a napot sokan úgy emlegették: „a nap, amikor a Google megölte a Web 2.0-t”. Tény és való, hogy a házilag, nehézkesen írt mashupok kora leáldozott. A MyMaps nem mashup és nem mashup készítő alkalmazás. Sokkal inkább egy testreszabható térképalkalmazás. A benne készült kódokat a Google mappleteknek hívja. A felhasználó a Google Maps API-jára épülve létrehozhat egy saját kódot (amely nem feltétlenül egy mashup), majd ezt publikussá teheti. A többi felhasználó pedig az eddigiekhez képest pofonegyszerűen adaptálhatja saját térképeire a mások által létrehozott kódokat. Ez lenne a Web 2.0 halála? Korántsem!

4.3 Célkitűzés

Célom, hogy a Google Maps szolgáltatásait kihasználva egy wifi hotspotokat gyűjtő mashupot hozzak létre. Magától értetődően két részre kell tagolódnia. Az elsőn a felhasználó egy térképen keresztül be tud majd jelenteni hotspotokat. Ezt egy ikon mozgatásával teheti majd meg, amelyet a kérdéses hotspot földrajzi helyére kell mozgatnia. Miután ezzel végzett, az ikonra kattintva egy formon keresztül el kell tudnia küldenie a hotspot SSID-jét[5], és a „Küldés" gombra kattintva bejelentenie az adott helyen található hotspotot. Természetesen nem csak az SSID-t küldjük a szerver felé, hanem a hely földrajzi koordinátáit is, amelyről a következő fejezetben lesz szó.

A szerveren a küldött adatokat egy PHP szkript fogja feldolgozni, és egy adatbázisban eltárolni. Itt természetesen gondoskodnunk kell arról is, hogy kétszer ne lehessen bejelenteni ugyanazt a hálózatot. Ehhez a hálózat SSID-jén kívül figyelnünk kell arra, hogy két azonos SSID-jű hálózat akkor tekinthető ugyanannak, ha egy meghatározott távolságon belül vannak. Erre azért van szükség, mert az emberek többsége még arra sem figyel, hogy a routerében megváltoztassa a gyártó által alapértékként beállított SSID-t. Ezért könnyen előfordulhat, hogy egymástól alig kétutcányira ugyanolyan (például „default") nevű hálózatokat találunk. Tudomásom szerint az SSID nem tartalmazhat ékezeteket, így ezek kezelésétől eltekintek.

A második komponensen a már bejelentett hotspotokat nézheti meg a felhasználó. Itt is egy-egy ikon jelzi az adott helyen lévő hotspotot, amelyre kattintva megnézhetjük az SSID-t. Természetesen a fentiekben felvázolt mashup rendkívül egyszerű, csak az alap funkciókra szorítkozik. Későbbiekben kézenfekvő bővítése lehet a két komponens egy térképen megjelenítése (bár ebben az esetben a használhatóság nagyméretű adatbázisnál megkérdőjelezhető), valamint az SSID-re keresés lehetősége. Esetleg egy adminisztrációs felület, ahol a bejelentett hotspotok státusza „működik, ellenőrizve"-re változtatható, módosítható, kitörölhető. Felhasználói oldalról érdekes információ lehet még, hogy milyen erősséggel fogták be az adott hálózatot.

[5] Egy vezetéknélküli hálózat SSID-je olyan, csak az angol ábécé karaktereit tartalmazó sztring, amellyel az adott hálózatra hivatkozunk

4.4 Földrajzi koordináták

Az emberi agy meghatározott földrajzi helyeket különböző támpontok alapján jegyez meg. Ahogy az utcán sétálunk, agyunk ilyen támpontokat keres, és dolgoz fel másodpercenként. Ez lehet akár egy különleges fa, egy buszmegálló, vagy éppen a helyi rendőrkapitányság. Ha egy meghatározott helyre akarunk eljutni, azt általában a címe alapján tesszük, ami egyértelműen beazonosítja a keresett helyet. De mi van az olyan nagy kiterjedésű helyekkel, mint például egy nemzeti park? Vagy bármi olyan hely, ahol a megszokott címzést nem lehet alkalmazni. Ilyen helyzetekben segítenek a földrajzi koordináták.

A földrajzi koordináták 2 komponensből állnak, egy vízszintes és egy függőleges információból. A földrajzi szélesség (latitude) jelenti a horizontális komponenst, az Egyenlítőtől való távolságot északra vagy délre. A földrajzi hosszúság (longitude) pedig a greenwichi kezdőkörtől (7. ábra) mért távolságot jelenti keleti vagy nyugati irányban. Mindkettőt fokokban, percekben, és másodpercekben mérik. A kettő kombinációjával bármely földrajzi hely pontosan meghatározható bárhol a Földön. Fontos kiegészítő információ a szemlélő helyzete a térképen, hiszen hiába vagyunk egy meghatározott helyén a Földnek, az, hogy merre nézünk, befolyásolja azt, hogy mit látunk.

7.ábra

A greenwichi kezdőkör

Ahhoz, hogy megértsük a Google Maps lényegét, alapvető szemléletváltásra van szükségünk. Amikor fotót készítünk például a parlament épületéről, akkor azt a nevet adjuk neki, hogy „Parlament”. Egy csomó lényeges információ elveszik. Hol készült a kép? Merre nézett a fényképész? Mikor készült? Ha ezt most levetítjük a földrajzi helyekre, akkor be kell látnunk, hogy itt is lényeges információkat vesztünk, ha nem mondjuk meg, hogy hol van, honnan és mikor szemléljük, merre nézünk.

4.5 Hotspotok bejelentése

4.5.1 Előkészületek

Bármiféle fejlesztés előtt otthonosan kell mozogni a Google Maps felületén, tisztában kell lenni az alapvető fogalmakkal. A fejlesztés során nem mappletet fogok elkészíteni, nem fogom kihasználni a MyMaps nyújtotta előnyöket. A http://www.google.com/apis/maps/signup.html oldalon kulcsot kell igényelni ahhoz a domainhez, amire a térkép fog kerülni. A kulcs semmilyen más domainhez nem lesz jó, amennyiben más helyre költöztetjük az oldalt, új kulcsot kell igényelni.

4.5.2 Keret

Teljes képernyős térkép létrehozása a célom, ehhez a következő keret-HTML szükséges:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
        <head>
                <meta http-equiv="content-type"
                content="text/html; charset=ISO-885-2"/>
                <title>Google Maps JavaScript API Example</title>
                <style type="text/css">
                        html, body {
                                height: 100%;
                                overflow: hidden;
                        }
                        body {
                                background-color: white;
                                font-family: Arial, sans-serif;
                                margin: 0;
                        }
                        #map {
                                height: 100%;
                        }
```

```
                </style>
                <script src="http://maps.google.com/maps?file=api&amp;v=2
                &amp;key=a_kapott_kod" type="text/javascript"></script>
                <script type="text/javascript">
                        //<![CDATA[
                        function load() {
                                if (GBrowserIsCompatible()) {
                                        //tényleges kód
                                }
                        }
                        //]]>
                </script>
        </head>
        <body onload="load()" onunload="GUnload()">
                <div id="map" style="width: 500px; height: 300px"></div>
        </body>
</html>
```

A tényleges kód a script tag-ek közé fog kerülni. Természetesen még sok egyéb kiegészítés is belevihető, de ez egy minimalista megoldás.

4.5.3 Az alap térkép létrehozása

Első lépés egy térkép objektum létrehozása, erre a

```
var map=new GMap2(document.getElementById("map"));
```

kódsor szolgál. Elhelyezését tekintve a kódban rögtön a //tényleges kód után következik. A térképnek szükséges egy kezdő pozíciót beállítani. Mivel a debreceni hotspotokat gyűjtjük, ezért kiinduló pontnak tökéletes a Kálvin tér. Meg kell még mondanunk, hogy a térkép mennyire legyen ráközelítve a kiindulási pontra. A 17-es ráközelítés tökéletes választás, ez a kontrollokkal elérhető maximum. A kiindulási pont megadásához egy földrajzi szélességet és hosszúságot tárolni képes objektumot hozunk létre, az osztály neve GLatLng. A szükséges kódsor tehát a térkép objektum létrehozása után: map.setCenter(new GLatLng(47.532669, 21.625091), 17);. A térképen még létre kell hoznunk egy nagyító-, és mozgató eszközt, a map.addControl(new GLargeMapControl()); utasítással. Ez a baloldalon felülre helyezi el a kontrollt, de paraméterezéssel máshová is elhelyezhető a térképen.

4.5.4 Ikon létrehozása

A térképeken megjelenő ikonokat a Google Maps markereknek hívja. A marker egy kisméretű ikon, amely a térképen precízen képes kijelölni egy pontot. Sok beépített marker létezik, de minden további nélkül definiálhatunk sajátot. A markerekre rá lehet kattintani,

ekkor egy buborékablak jelenik meg, melybe tetszőleges információ (akár kép, videó is) helyezhető. Maga a kattintás egy esemény, amire eseménykezelőt kell írnunk. Egy marker ikont egy GIcon objektum reprezentál, amelynek az attribútumait megfelelő módon be kell állítanunk. Az image attribútum maga az ikon, egy .png kép (az átlátszóság miatt), az iconSize pedig az ikon mérete, értékéül GSize-t kell adnunk. Minden ikonnak kell lennie egy horgonypontjának (anchor), ez az a pontja, amely a precíz pozíciót fogja kijelölni (például egy célkeresztnél anchor pont a kereszt közepe). Ezt az iconAnchor attribútum beállításával adhatjuk meg. Be kell még állítanunk az infoWindowAnchor attribútumot, amely a buborékablak horgonypontját adja meg az ikon bal felső sarkához képest, értéke szintén GPoint. Egy ikon objektum elkészítése tehát például így néz ki:

```
var wifiIcon = new GIcon();
wifiIcon.image = " icon.png";
wifiIcon.iconSize = new GSize(31, 24);
wifiIcon.iconAnchor = new GPoint(15, 23);
wifiIcon.infoWindowAnchor = new GPoint(15, 1);
```

Ezen kívül még maradt beállítási lehetőség, a fenti példában nem éltünk például az ikonhoz tartozó árnyék létrehozásának a lehetőségével. A felhasznált ikon a következőképpen néz ki:

8. ábra
A wifi ikon

31 pixel széles, és 24 pixel magas, térképes megjelenítésre ideális méretű. Horgonypontnak a három lába közül a középsőt választottam. A buborékablak horgonypontja pedig az ikon közepére, egy pixellel fölé fog esni.

Természetesen az ikon objektum létrehozása önmagában még nem elég, markert kell belőle készíteni. Ezt a

```
var point = new GLatLng(47.532669, 21.625091);
var marker=new GMarker(point,{icon:wifiIcon, draggable: true, bouncy: true});
```

sorokkal tehetjük meg. Ebben az esetben a marker a Kálvin térre fog kerülni (mint ahogy már korábban láthattuk, a térkép közepének beállításánál), ikonja az előbbiekben létrehozott ikon lesz. Természetesen mozgathatónak kell lennie (draggable), és a bouncy true-ra állításával látványos leesési effektet kapcsolhatunk be. Az így elkészült ikont rá is kell helyezni a térképre, mint overlay-t, a map.addOverlay(marker); utasítással.

4.5.5 A térkép mozgatásának és a markernek a kérdése

A térkép tényleges használata során, mivel a térkép mozgatható, kézenfekvő igény, hogy a marker kövesse a térkép mozgását. Hiszen ha a marker kikerül az általunk látott térképrészről, lehetséges, hogy nem fogjuk megtalálni. Erre a következő megoldást alkalmazzuk: ha a marker a térkép mozgatása során kikerül a látható térképrészről, akkor automatikusan át fog kerülni az új térképrész közepére. Egy eseménykezelőt kell írni, amely a térkép objektum moveend eseményre reagál (a mozgatás befejezésekor hívódik meg). A szükséges kód:

```
GEvent.addListener(map, 'moveend', function(overlay, point) {
        var bounds = map.getBounds();
        if(!bounds.contains(marker.getPoint()))
                marker.setPoint(bounds.getCenter());
});
```

4.5.6 Kattintás a markerre

A markerre kattintáskor meg kell tehát jelenítenünk egy formot, amelyben el tudjuk küldeni a hotspot SSID-jét, és rejtett információként a földrajzi szélességet és hosszúságot. Szintén egy eseménykezelőt kell írnunk, ami a marker click eseményére reagál. Magát a formot JavaScript kód fogja létrehozni.

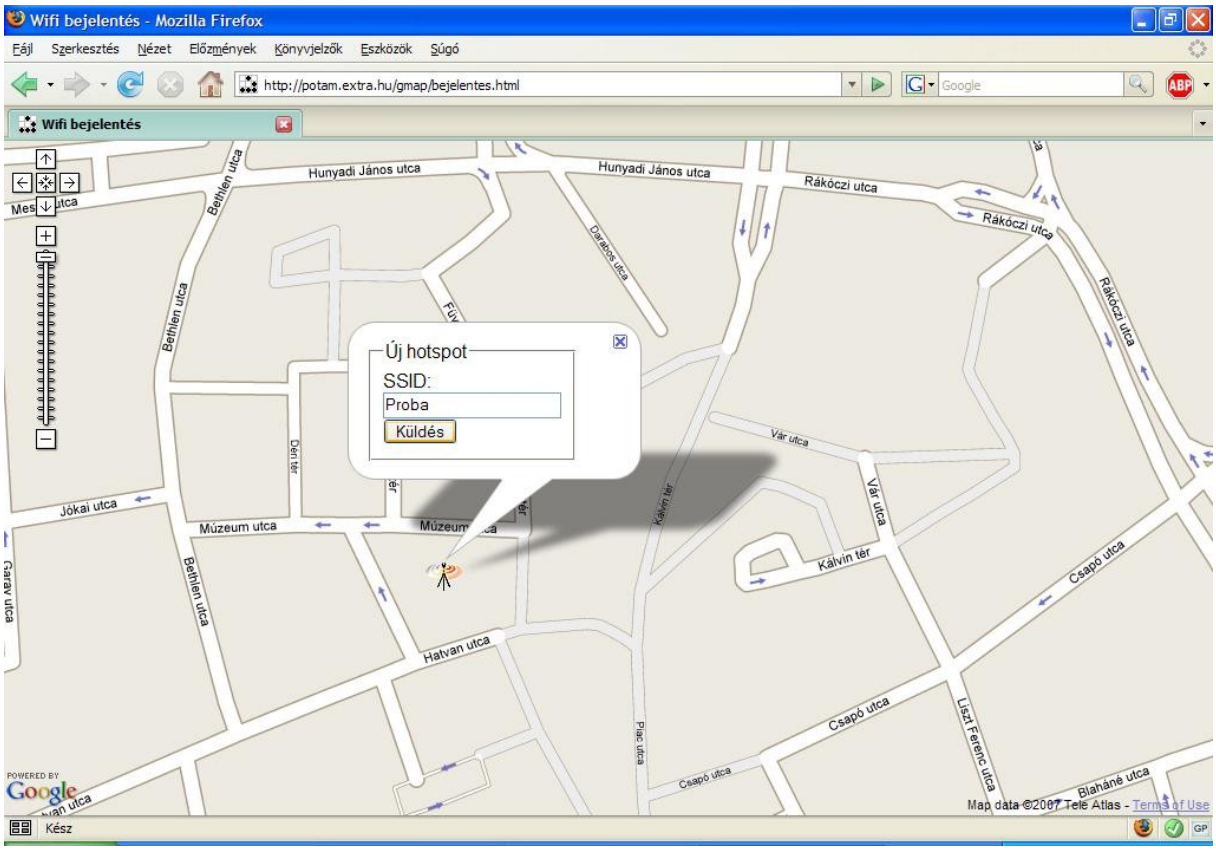
```
GEvent.addListener(marker, "click", function() {
        var inputForm=document.createElement("form");
        var longitude=marker.getPoint().lng();
        var latitude=marker.getPoint().lat();
        inputForm.setAttribute("id","wifiform");
        inputForm.innerHTML='<fieldset style="width:150px;">'
        + '<legend>Új hotspot</legend>'
        + '<label for="ssid">SSID:</label>'
        + '<input type="text" id="ssid" style="width:98%;" />'
        + '<input type="hidden" id="longitude" value="'+longitude+'" />'
```

```
        + '<input type="hidden" id="latitude" value="'+latitude+'" />'
        + '<input type="button" value="Küldés" onclick="saveWifi(); return false;" />'
        + '</fieldset>';
        marker.openInfoWindow(inputForm);
});
```

Látható, hogy milyen módon tudjuk egy marker földrajzi koordinátáit lekérdezni, amelyek ezután a form két rejtett elemének értékei lesznek. Megjegyzendő, hogy a buborékablakba tetszőleges HTML kód helyezhető. A marker helyzetét ténylegesen a saveWifi függvény fogja elmenteni, amely a GXmlHttp osztály segítségével aszinkron kérést fog indítani a wifi.php szkript felé, amely majd a kapott adatokat ellenőrzi, és elmenti adatbázisba. A wifi.php kódját nem részletezzük, a szükséges JavaScript függvény:

```
function saveWifi()
{
        var longitude = document.getElementById("longitude").value;
        var latitude = document.getElementById("latitude").value;
        var ssid = document.getElementById("ssid").value;
        var getVars = "?ssid=" + ssid + "&longitude=" + longitude + "&latitude=" + latitude ;
        var request = GXmlHttp.create();
        request.open('GET', 'wifi.php' + getVars, true);
        request.onreadystatechange = function() {
                if (request.readyState == 4) {
                        response = request.responseText;
                        marker.openInfoWindowHtml(response);
                }
        }
        request.send(null);
        return false;
}
```

Korábban említettem, hogy két azonos nevű hotspotot akkor tekintünk azonosnak, ha elég közel vannak egymáshoz. Mivel egy átlagos router 100 méteren belül képes sugározni, ezért ez a távolság ideális. Két földrajzi fok közötti távolság azonban nem mindig ugyanakkora, ugyanis a Föld nem tökéletes gömb. Én az átlaggal számolok, tehát 111 kilométerrel. Így a 100 méter 0,0009009009 fokot jelent. Ezzel elkészültem az első résszel, a felhasználó képes a Google Maps térképen keresztül hotspotot bejelenteni (9. ábra). Működő változat megtekinthető itt: http://potam.extra.hu/gmap/bejelentes.html



9.ábra

Az elkészült hotspot bejelentő felület

4.6 Meglévő hotspotok lekérdezése

4.6.1 Szerver oldal

A beküldött hotspotokat MySQL adatbázis tárolja, a tárolt adatok az ssid, a longitude, és a latitude. Szerver oldalon szükség van egy getwifi.php fájlra, amely az eddig tárolt hotspotokat fogja szabályos XML formátumba visszaadni. A választott XML szerkezet a következő:

```
<hotspots>
        <hotspot>
                <ssid>data</ssid>
                <longitude>data</longitude>
                <latitude>data</latitude>
        </hotspot>
        […]
</hotspots>
```

Az ezt előállító php szkript szintén triviális, számtalan módszerrel megírható, akár XML framework-öt is felhasználva. Ezt az XML fájlt fogja majd a megjelenítésért felelős JavaScript kód aszinkron módón lekérni, és felépíteni belőle a markereket.

4.6.2 Kliens oldal

Kliens oldalon természetesen szükségünk van a bejelentő oldalon ismertetett HTML keretre. Létre kell hoznunk a térkép objektumot, beállítani a középpontját a Kálvin térre, és hozzáadni a nagyító kontrollt. Létre kell hoznunk még a `wifiIcon` objektumot, ugyanis minden ikon ezt fogja használni, fölösleges egyenként újra és újra létrehozni, elég egyszer a kód elején. Ha ez kész, akkor `GXmlHttp` objektummal le kell kérnünk a getwifi.php által generált XML fájlt:

```
var request = GXmlHttp.create();
request.open('GET', 'getwifi.php', true);
```

majd a kérés elküldése előtt meg kell adnunk a függvényt, ami fel fogja dolgozni az XML fájlt. Először egy tömbben el kell tárolnunk a hotspot node-okat, hogy aztán majd egyenként egy ciklusban kinyerjük belőlük az adatokat:

```
var xmlDoc = request.responseXML;
var markers = xmlDoc.documentElement.getElementsByTagName("hotspot");
```

A `markers` tömb méretét a `markers.length` adja meg, a tömb indexelése nulláról indul.

Következő feladat kinyerni az aktuális hotspot node-ból a gyermekeiben tárolt adatokat. A szükséges kód:

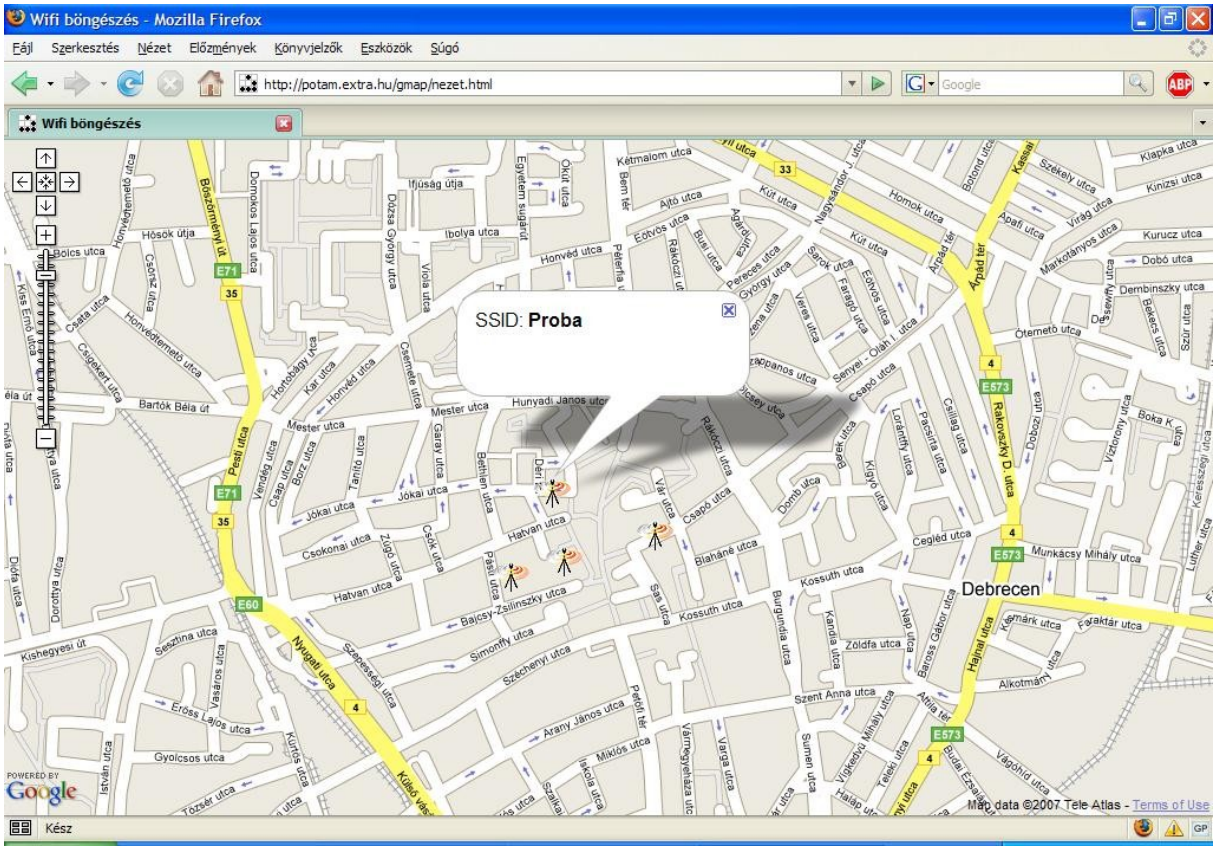
```
longitude=parseFloat(markers[i].getElementsByTagName("longitude")[0].firstChild.data);
latitude=parseFloat(markers[i].getElementsByTagName("latitude")[0].firstChild.data);
ssid = markers[i].getElementsByTagName("ssid")[0].firstChild.data;
```

Létre kell hoznunk egy pont objektumot a kinyert longitude és latitude értékekből: `point = new GLatLng(latitude,longitude);`, majd a kapott pontban létrehozni egy markert, és hozzáadni a térképhez:

```
var marker = createMarker(point, ssid, wifiIcon);
map.addOverlay(marker);
```

Nagyon fontos a createMarker függvény használata (amelyet később írunk meg). Ugyanis a markerekhez click eseménykezelőt is szeretnénk hozzárendelni, amely meghíváskor buborékablakban megmutatja az eltárolt SSID-t. Nem hozhatjuk létre itt, ebben a függvényben a markert, new GMarker... kóddal, mert akkor az ezután logikusan következő lépésben létrehozandó click eseménykezelő a markerhez mindig egyetlen (mégpedig a legutolsó) markerhez fog kötődni. A megoldás a marker létrehozásának kiemelése egy külön függvénybe, így az eseménykezelés is megfelelő lesz. A szükséges kód:

```
function createMarker(point, ssid, wifiIcon) {
        var marker = new GMarker(point,{icon:wifiIcon, draggable: false});
        var html = "SSID: <b>" + ssid + "</b>";
        GEvent.addListener(marker, 'click', function() {
                marker.openInfoWindowHtml(html);
        });
        return marker;
}
```



10. ábra

Regisztrált hotspotok megtekintése

Működő változat megtekinthető itt: http://potam.extra.hu/gmap/nezet.html . Ennek a technikának nagy hátránya, hogy az egész térképre egyszerre helyezi el a markereket. Kézenfekvő optimalizálás lenne, egy tömbben összegyűjteni a még ki nem rajzolt markereket, és csak az aktuális térképrészletre kerülő, még ki nem rajzolt markereket elhelyezni. Kérdés ebben az esetben, hogy mekkora a tömb bejárása miatti időveszteség okozta felhasználói-élmény romlás.

4.7 Összefoglalás

A fenti leírás alapján sikerült egy olyan wifi-gyűjtő mashup létrehozása, amely alkalmas kisszámú (<200) hotspot összegyűjtésére, és vizuális megjelenítésére. Tényleges használata előtt még további fejlesztésre szorul. Szükség van a hotspot SSID-jének bejelentése mellett a vételi erősség megadására is. Regisztrációs rendszerben működve a felhasználó által bevitt hotspotok későbbi módosítására is célszerű lehetőséget adni, beleértve a törlést is. Létre kell hozni egy adminisztrációs felületet, ahol az adminisztrátor „ellenőrzött” státuszúvá teheti a bejelentett hotspotot. Megjelenítési felületen természetes igény lehet a hotspotok szűkítésére. Például el lehet helyezni oldalt egy listát az utolsó 10 bevitt hotspotról. SSID-re történő keresést lehet készíteni. Érdekes lehet az a megoldás, hogy a felhasználó a térképre kattintva, a kattintás pontjától számított x méteren belül keres hozzá közeli hotspotot, esetleg az odajutásra is kíváncsi. Ugyanakkor a létrehozott kód rengeteg helyen felhasználható, tetszőleges objektumok gyűjtésére alkalmas, akár különböző típusúakéra is. Étteremhálózatoktól kezdve ingatlankereskedésekig, rengeteg specifikus területre ültethető át kisebb-nagyobb módosításokkal.

Irodalomjegyzék

[1] Dave Crane, Eric Pascarello: Ajax in Action. Manning, 2005.

[2] Filip Chereches-Tosa [et al.]: AJAX and PHP: Building Responsive Web Applications. Packt publishing, 2006.

[3] Martin C. Brown: Hacking Google Maps and Google Earth. Wiley Publishing, 2006.

[4] Web 3.0, avagy a jövő. 2006.11.02. In http://webizen.hu/2006/11/02/web-30-avagy-a-jovo/ . Accessed 2007.10.23.

[5] Brian Getting: Basic Definitions: Web 1.0, Web. 2.0, Web 3.0. 2007.04.18. In http://www.practicalecommerce.com/articles/464/Basic-Definitions-Web-10-Web-20-Web-30/ . Accessed 2007.10.23.

[6] Bőgel György: Web 3.0, az újabb hívószó. 2007.03.22. In http://www.sg.hu/cikkek/51225 . Accessed 2007.10.23.

[7] Ajax. In http://hu.wikipedia.org/wiki/Ajax . Accessed 2007.10.23.

[8] A Google szolgáltatásainak listája. In http://hu.wikipedia.org/wiki/A_Google_szolg%C3%A1ltat%C3%A1sainak_list%C3%A1ja . Accessed 2007.10.23.

[9] Google cheat sheet. In http://www.feedsforme.com/google/?cheatsheet&page=2 . Accessed 2007.10.23.

[10] Google Maps API. In http://www.google.com/apis/maps/documentation/ . Accessed 2007.10.23.

[11] Jesse James Garrett: A New Approach to Web Applications. 2005.02.18. In http://www.adaptivepath.com/ideas/essays/archives/000385.php . Accessed 2007.10.28.

[12] pyxy-gallery. In http://fennecfoxen.org/pyxy/gallery . Accessed 2007.10.28.

[13] Lightbox JS. In http://www.huddletogether.com/projects/lightbox/ . Accessed 2007.10.28.

[14] minishowcase. In http://minishowcase.frwrd.net/ . Accessed 2007.10.28.